EXTERNAL REFERENCE SPECIFICATION

for

Miscellaneous Routines Interface

Submitted: W. R. Bayer

Approved: _____
J. J. Krautbauer

_____
P. W. Haynes

_____
J. R. Ruble

DISCLAIMER:

This document is an internal working paper only. It is subject to change, and does not necessarily represent any official intent on the part of CDC.

REVISION DEFINITION SHEET

REV	DATE	DESCRIPTION
1	02/20/79	Original Release.
2	03/07/79	NOS170 related disclamer, rearrangement to deemphasize NOS170, clarification of nuances.
3	05/24/79	Miscellaneous clarifications, added zutpaqr, zn7prdr, zutpifw, zutpw2h, zutpw2o, zutpmsg descriptions.
4	10/22/79	Added zutpcap, zutpirs, zosptwm, terminal interrupt processing and overlay loading descriptions. Name change: PASCAL-X to CYBIL.
5	01/22/80	Added zutfboj, zutpaft, zutpask, zutpcaa, changes to terminal interrupt, Simulator checkpoint routines, miscellaneous changes.
6	07/16/80	Document format changes. Obsoletes all previous versions.

1.0 INTRODUCTION

The programming language used in this implementation is CDC CYBIL. The details of the interface are defined in terms of CYBIL structures.

1.0 INTRODUCTION
1.1 SCOPE OF DOCUMENT

1.1 SCOPE OF DOCUMENT

This document is one of a set of documents describing portions of the interfaces which are part of the SES Utility Library. A seperation has been made to simplify documentation efforts and to exemplify the natural modularity.

This document contains information necessary for the understanding and use of Miscellaneous Routines available through the Software Engineering Services Utility Library (SESUL).

1.2 ASSOCIATED DOCUMENTS

The following documents may be referenced in part to obtain a more complete understanding of the origin, uses and nomenclature associated with Miscellaneous Routines.

NOS/VE ERS

Language Specification for CDC CYBIL(ARH2298)

ERS for CYBIL I/O (ARH2739)

CYBER 180 System Interface Standard (S2196)

SES Procedure Writers GUIDE (ARH2894)

SES User's Handbook (ARH1833)

NOS Reference Manual Volume 2 (pub. no. 60445300)

System Command Language (SCL) ERS (SES Internal)

Input Output Control (IOC) ERS (SES Internal)

Message Generator ERS (SES Internal)

Command Processor (CP) ERS (SES Internal)

SES Processor ERS (SES Internal)

1.3 NAMING CONVENTIONS

The following naming conventions have been imposed upon the Miscellaneous Routines and SES Utility Library interface routines in general.

Decknames are of the form Zpcyxxx where:

Z	universal SES identifier
pc	two character interface identifier OS NOS/VE operating system compatible PM NOS/VE program management compatible N7 directly related to or dependent upon a feature of NOS 170 UT utility (none of the above)
y	type of deck I Compass module (Ident) P CYBIL procedure reference C CYBIL constant declaration T CYBIL TYPE declaration V CYBIL variable declaration M CYBIL module F CYBIL function
xxx	three characters representing the abbreviated descriptive name of the deck (suggestion is first characters of the words composing the descriptive name).

1.0 INTRODUCTION
1.3 NAMING CONVENTIONS

Procedure names are of the form pcp$xxxxxxxxxxxxxxx where

pc	two character product identifier
p	CYBIL procedure identifier
xxxxxxxxxxx	a meaningful, descriptive name of procedure. All procedure names are limited to 31 characters (including prefix).

1.4 DISCLAIMER RELATED TO NOS 170

Procedures using "N7" as an interface identifier have direct NOS 170 dependencies. It is assumed the user of "N7" procedures has indepth experience and knowledge of NOS 170 (the user of "N7" must know what he is doing).

Any compatibility between "N7" procedures and future NOS/VE supplied procedures is purely coincidental. Therefore, use the "N7" procedures at your own discretion.

1.5 MISCELLANEOUS ROUTINES USAGE

All procedures described in this document are available for use. Common decks are made available by specifying the "CYBCCMN" keyword on the SES.GENCOMP procedure. The binaries are available for linking by specifying the "CYBCLIB" keyword on the SES.LINK170 procedure.

Note that more current copies of miscellaneous routines are generally made available in the "SSS" catalog as a function of prerelease while a release version is found on the "SES" catalog. A complete summary of new decks added during prerelease may be obtained from your local SES representative.

2.0 MISCELLANEOUS ROUTINES DESCRIPTION

The routines classified as miscellaneous are a disjunct set. They perform services such as CYBIL to NOS interfaces, conversion of data, file manipulations, system utility operations, CYBIL program control, and string manipulations.

Any commonality among these routines lies in the fact that they are self contained primitives. It is intended for the SES Utility Library that the miscellaneous routines are to be an "on-going", growing group.

If you, the reader, have additional routines which are useful make them known. It is the the intent of this document to consolidate individual efforts.

2.0 MISCELLANEOUS ROUTINES DESCRIPTION
2.1 OBJECTIVES OF MISCELLANEOUS ROUTINES

2.1 OBJECTIVES OF MISCELLANEOUS ROUTINES

The objectives of the Miscellaneous Routines are:

1) Provide a documented base of diversified routines which may be added to when a need arises,

2) Provide a group of CYBIL routines each serving a unique purpose,

3) Provide general purpose, independent routines (routines may have basic NOS170 or file dependencies).

2.2 PHILOSOPHY OF MISCELLANEOUS ROUTINES

When there is a task to be done provide a routine to do it. The routine is an "end condition" or a function that has no lateral dependencies. It can have limited upward or downward dependencies if necessary. Each routine exhibits the characteristic of filling a unique purpose whose scope is limited to that routine.

3.0 MISCELLANEOUS ROUTINES INTERFACES

3.1 DESCRIPTION

The Miscellaneous Routine interfaces consist solely of a set of procedure interfaces. They are grouped in terms of what they interface to or what data they operate upon. This set is intended to be in a constant state of growth as coding within the SES Tools Group proceeds and new requirements arise.

A summary of each procedure is given along with the procedure reference in the form of a common deck. Where necessary, to further explain parameters, other common decks are included. All information included is intended to be self explanatory.

An appendix is included with a list of all common decks that exist as *calic within the procedure reference common decks. This alphabetic appendix lists contents that are composed of TYPE and CONST information.

Since it is not intended to rewrite any of the available NOS 170 manuals, any procedures which require knowledge of specific areas of the manuals will reference them. The user should consult that document. Also see "N7" disclaimer in this document.

3.2 PROCEDURES

3.2.1 DATA CONVERSION PROCEDURES

3.2.1.1 Capitalize_String

The following procedure capitalizes the alphabetic characters in a string.

```
{ZUTPCAP    capitalize a string

  PROCEDURE [XREF] utp$capitalize_string ALIAS 'zutpcap' (VAR char_string:
    string ( * ));
```

3.2.1.2 Display_Code_Name_to_CYBIL_String

The purpose of this procedure is to convert a standard NOS 170 seven character display code name to a seven character CYBIL string which is left justified and blank filled.

```
*calic zuttdcn

{ ZUTPDNS    Converts NOS 170 7 char. disp. code name to CYBIL string. }

  PROCEDURE [XREF] utp$convert_dc_name_to_string ALIAS 'zutpdns' (dc_name:
    utt$dc_name;
    VAR result_string: string (7);
    VAR result_length: 0 .. 7);
```

3.2.1.3 CYBIL_String_to_Display_Code_Name

This procedure converts CYBIL string to a standard NOS 170 seven character display code name. The conversion proceeds until either seven characters have been processed or a blank (space) character is encountered. The resulting name is blank filled for each character short of 7.

```
*calic zuttdcn

{ ZUTPSDN    Converts CYBIL string to NOS 170 7 char. disp. code name. }

  PROCEDURE [XREF] utp$convert_string_to_dc_name ALIAS 'zutpsdn'
    (source_string: string ( * );
    VAR dc_name: utt$dc_name);
```

3.2.1.4 CYBIL_String_to_Display_Code_File_Name

This procedure converts an adaptable CYBIL string to a standard NOS 170 seven character display code file name. The conversion proceeds until either seven characters have been processed or a non-alphanumeric character is encountered. The resulting display code file name is left justified, zero filled (e.g., for use in FET).

```
*callc zuttdcn

{ ZUTPSFN    Converts adapt. CYBIL string to NOS 170 disp. code file
{name. }

  PROCEDURE [XREF] utp$convert_string_to_file_name ALIAS 'zutpsfn'
    (source_string: string ( * );
    VAR dc_file_name: utt$dc_name);
```

3.2.1.5 CYBIL_String_to_Display_Code_String

The purpose of this procedure is to convert a CYBIL string to a display code string. The length of the display code string is in terms of words. Both a word index and character position within the word are input to and updated by the procedure. Conversion stops when the display code string is filled or the CYBIL string is exhausted. If the EOL parameter is TRUE when this procedure is called and there is room for the entire CYBIL string and an end-of-line in the display code string, then an end-of-line is generated in the display code string following the converted CYBIL string. If the EOL was generated, then the EOL parameter is set to TRUE, otherwise it is set false. An EOL is defined (in a display code string) as a right justified field of 12 to 66 bits of zeros.

```
*callc zoststr

{ ZUTPS2D    Converts CYBIL string to display code string. }

  PROCEDURE [XREF] utp$convert_string_to_dc_string ALIAS 'zutps2d'
    (encoding: (utc$ascii64, utc$ascii612);
    VAR dc_string: array [ * ] OF packed array [0 .. 9] OF 0 .. 3f(16);
    VAR dc_string_word_index: integer;
    VAR dc_string_char_index: 0 .. 9;
    source_string: string ( * );
    VAR source_index: ost$string_index;
    VAR eol: boolean);
```

3.2.1.6 Display_Code_String_to_CYBIL_String

The purpose of this procedure is to convert a display code string to a CYBIL string. The length of the display code string is in terms of words. Both a word index and a character position within the word are input to and updated by the procedure. Conversion stops when: 1) the CYBIL string is filled, 2) the display code string is exhausted, or 3) an EOL (end-of-line) is found. An EOL is defined (in a display code string) as a right justified field of 12 to 66 bits of zeros.

```
*callc zoststr

{ ZUTPD2S    Converts display code string to CYBIL string. }

  PROCEDURE [XREF] utp$convert_dc_string_to_string ALIAS 'zutpd2s'
    (encoding: (utc$ascii64, utc$ascii612);
    VAR dc_string: [READ] array [ * ] OF packed array [0 .. 9] OF 0 ..
      63;
    VAR dc_string_word_index: integer;
    VAR dc_string_char_index: 0 .. 9;
    VAR result_string: string ( * );
    VAR result_length: ost$string_length;
    VAR eol_found: boolean);
```

3.2.1.7 Integer_to_String

The purpose of this procedure is to convert an integer to its string representation in the specified radix. If the integer is negative, the leftmost character of the resulting string is a '-'. The string containing the integer's representation left justified and the length of the representation is returned. This length is zero if the string is too small to represent the integer. This procedure can handle integers with values in the range -(2**59-1) .. 2**59-1.

```
*calic zoststr

{ ZUTPI2S    Converts integer to string rep. in specified radix. }

  PROCEDURE [XREF] utp$convert_integer_to_string ALIAS 'zutpi2s' (VAR
    result_string: string ( * );
    VAR result_length: ost$string_length;
    source_integer: integer;
    radix: 2 .. 16);
```

3.2.1.8 Integer_to_Right_Justified_String

The purpose of this procedure is to convert an integer to its string representation in the specified radix. The resultant string contains the string representation of the integer right_justified and zero filled. If the integer is negative, the leftmost character of the resulting string is a '-'. Should the procedure fail, a boolean is set false. Conditions causing failure are overflow (result string too small) and an invalid source string. This procedure can handle integers with values in the range -(2**59-1) .. 2**59-1.

```
*CALLC zoststr

{ ZUTPIRS    procedure to convert integer to right justified string

PROCEDURE [XREF] utp$convert_integer_to_rjstring ALIAS 'zutpirs'
    (VAR result_string : string (*);
     VAR conversion_okay : BOOLEAN;
         source_integer: integer;
         radix: 2 .. 16);
```

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.2.1.9 String to Integer

3.2.1.9 String_to_Integer

The purpose of this procedure is to convert the string representation of an integer to an integer value. The procedure begins its examination of the source string at the position specified by the source index. That index is incremented by one for each character of the string that is used. The integer may be preceeded by a sign (+ or -). The first character of the integer must be a decimal digit, however, subsequent characters of the integer may be decimal digits or letters (case ignored) A through F (representing hex digits 10 through 15). The integer itself can optionally be immediately followed by a radix specification (unsigned integer 2 through 16 with no leading zeros and enclosed in parentheses). In the absence of a radix specification, 10 is assumed. The radix value must be larger than the largest digit value in the integer. This procedure can handle integers with values in the range -(2**59-1) .. 2**59-1.

```
*callc zoststr

{ ZUTPS2I    Converts string rep. of integer to integer value. }

  PROCEDURE [XREF] utp$convert_string_to_integer ALIAS 'zutps2i' (VAR
    source_string: [READ] string ( * );
    VAR source_index: ost$string_index;
    VAR result_integer: integer;
    VAR conversion_worked: boolean);
```

3.2.1.10 Character_Translation_(Conversion)_Structure

The following are used as translation tables in the conversion to/from ascii. Conversion of ascii to ascii612, ascii612 to ascii, ascii to ascii64, and ascii64 to ascii are available.

```
{ ZUTVCTT    Translation table used in conversion to/from ascii. }

  VAR
    utv$convert_ascii_to_ascii612 ALIAS 'cvas612': [XREF, READ] array
      [char] of packed record
      case long: boolean of
      = FALSE =
        f1: set of 1 .. 53,
        ch: 0 .. 3f(16),
      = TRUE =
        f2: set of 1 .. 47,
        escape_ch: 0 .. 3f(16),
        follower_ch: 0 .. 3f(16),
      casend,
    recend,
    utv$convert_ascii612_to_ascii ALIAS 'cv612as': [XREF, READ] array [0
      .. 3f(16)] of packed record
      case escape: boolean of
      = FALSE =
        f1: set of 1 .. 51,
        ch: char,
      = TRUE =
        f2: set of 1 .. 41,
        conv: ^array [0 .. 3f(16)] of char,
      casend,
    recend,
    utv$convert_ascii_to_ascii64 ALIAS 'cvasc64': [XREF, READ] array
      [char] of 0 .. 3f(16),
    utv$convert_ascii64_to_ascii ALIAS 'cv64asc': [XREF, READ] array [0
      .. 3f(16)] of char;
```

3.2.1.11 Word_to_Hexadecimal_String

The purpose of this procedure is to produce the hexadecimal interpretation of the contents of a word.

```
{ ZUTPW2H    Produces hex interpretation of contents of word. }

  PROCEDURE [XREF] utp$word_to_hexadecimal_string ALIAS 'zutpw2h'
    (pointer_to_word: ^cell;
    VAR hexadecimal_string: string (15));
```

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.2.1.12 Word to Octal String

3.2.1.12 Word_to_Octal_String

The purpose of this procedure is to produce the octal interpretation of the contents of a word.

```
{ ZUTPW20    Produces octal interpretation of contents of word. }

  PROCEDURE [XREF] utp$word_to_octal_string ALIAS 'zutpw2o'
    (pointer_to_word: ^cell;
    VAR octal_string: string (20));
```

3.2.2 FILE PROCEDURES

3.2.2.1 Acquire_a_File

This procedure provides a "high-level" interface to the facility made available via procedure n7p$acquire_file (see the description of that procedure for details).

```
*calic zn7ppfm
*calic zuttaqr

{ ZUTPAQR    Interfaces facility made avail. to localize a file. }

  PROCEDURE [XREF] utp$acquire_file ALIAS 'zutpaqr' (local_file_name:
    string ( * );
    permanent_file_name: string ( * );
    user_name: string ( * );
    password: string ( * );
    pack_name: string ( * );
    mode: n7t$pfm_modes;
    request: utt$acquire_request_codes;
    VAR response: utt$acquire_response_codes);
```

3.2.2.2 File_Assigned_to_Job?

The purpose of this procedure is to determine if a file is assigned (local) to a job.

```
{ ZUTPIFL    Determines if file is assigned (local) to a job. }

  PROCEDURE [XREF] utp$is_file_local ALIAS 'zutpifl' (file_name: string
    ( * );
    VAR is_file_local: boolean);
```

3.2.2.3 Return_a_File

The purpose of this procedure is to remove the assignment of a file to the current job.

```
{ ZUTPRTF    Removes assignment of file to current job. }

  PROCEDURE [XREF] utp$return_file ALIAS 'zutprtf' (file_name: string (
    * ));
```

3.2.2.4 Rewind_a_File

The purpose of this procedure is to position a file at its beginning of information.

```
{ ZUTPRWF    Positions file at its Beginning Of Information. }

  PROCEDURE [XREF] utp$rewind_file ALIAS 'zutprwf' (file_name: string (
    * ));
```

3.2.2.5 Message About NOS 170 Permenant File

The purpose of this procedure is to issue an informative message to the dayfile concerning a permanent file. The format of the message is:

```
xxxxxxxx PFN=nnnnnnn UN=uuuuuuuu
```

Where xxxxxxxx is the supplied message string
nnnnnnn is the permanent file name (obtained from the FET)
uuuuuuu is the user name of the owner of the file (obtained from the FET).
If the user name field of the FET is 0, the UN= part of the message is omitted.

```
*calic zn7tfet

{ ZN7PPIM    Issues message to dayfile concerning permanent file. }

  PROCEDURE [XREF] n7p$pf_info_message ALIAS 'zn7ppim' (main_message:
    string ( * );
    VAR fet_with_pfn_and_un: n7t$fet);
```

3.2.2.6 Localize_File

The purpose of this procedure is to locate (acquire -- make local) the file specified by a FET. Searching for the file is restricted by the nature of the request code. If the file was local, it is rewound. If the file is accessed via PFM, both an attach and a get are attempted. If necessary a wait is done for the file to become not busy or for PFM to become not busy, providing the error processing bit in the FET is set. If the bit is set the procedure assumes that the erad field of the FET is set as well. The pfm_error_occurred parameter is set TRUE if PFM gives a response other than:

n7c$pfm_file_found,
n7c$pfm_file_not_found,
n7c$pfm_file_busy, or
n7c$pfm_pf_utility_active otherwise it is unaltered.

```
*calic zn7tfet
*calic zuttaqr

{ ZN7PAQR    Localizes a file specified by an FET. }

  PROCEDURE [XREF] n7p$acquire_file ALIAS 'zn7paqr' (VAR fet: n7t$fet;
    request: utt$acquire_request_codes;
    VAR response: utt$acquire_response_codes);
```

The acquire request and response codes are found on deck ZUTTAQR.

```
{ ZUTTAQR    Contains acquire request and response codes. }

  TYPE
    utt$acquire_request_codes = (utc$acquire_anywhere,
      utc$acquire_local_only, utc$acquire_permanent_only),
    utt$acquire_response_codes = (utc$acquire_not_found,
      utc$acquire_was_local, utc$acquire_was_indirect,
      utc$acquire_was_direct, utc$acquire_error);
```

3.2.2.7 Set_Record_Type

This procedure determines the name and type of a NOS 170 logical record from the first 64 words located in a working buffer. The procedure reference is found on common deck ZN7PSRT while type information for records is found on deck ZN7TSRT.

```
{ ZN7TSRT    Contains type information for records. }

  CONST { NOS 170 symbols for 'logical' record types }
    n7c$text = 0(16),
    n7c$pp = 1(16),
    n7c$cos = 2(16),
    n7c$rel = 3(16),
    n7c$ovl = 4(16),
    n7c$ulib = 5(16),
    n7c$opl = 6(16),
    n7c$oplc = 7(16),
    n7c$opld = 8(16),
    n7c$abs = 9(16),
    n7c$ppu = 0a(16),
    n7c$cap = 0e(16),
    n7c$proc = 10(16);
```

```
*calic zuttdnv

{ ZN7PSRT    Determines name and type of NOS 170 logical record. }

  PROCEDURE [XREF] n7p$set_record_type ALIAS 'zn7psrt' (ptr_to_record:
    ^cell;
    VAR record_name_and_type: utt$dc_name_and_value);
```

3.2.2.8 Is_File_Writable?

This procedure determines if a file is writable (e.g., attached in write mode) by the current job.

```
{ ZUTPIFW    Determines if file is writable by current job. }

  PROCEDURE [XREF] utp$is_file_writable ALIAS 'zutpifw' (file_name:
    string ( * );
    VAR is_file_writable: boolean);
```

3.2.2.9 Get_Directory_Record_from_Binary_File

The purpose of this procedure is to read, from a (binary) file, a directory record (type OPLD). The directory (if it exists) must be the last record in the file (optionally followed by an end_of_file mark. If no directory is found a NIL pointer is returned. If the directory is found, space is allocated for it in the system heap, the record descriptor entries are read into that space and a pointer to the space is returned.

```
*calic pxiotyp
*calic zn7tdir

{ ZN7PRDR    Reads from a binary file a directory record. }

  PROCEDURE [XREF] n7p$get_opld_directory ALIAS 'zn7prdr' (binary_file:
    file;
    VAR opld_directory_pointer: ^n7t$opld_directory);
```

3.2.2.10 Assign_Legible_File_to_Terminal

This procedure assigns a legible file to a terminal. The assignment should be done before the file is opened. It is the users responsibility to open and close the file. Any problems encountered result in the file_assigned variable set false.

```
{ ZUTPAFT    Assign file to a terminal }

  PROCEDURE [XREF] utp$assign_file_to_terminal ALIAS 'zutpaft' (file_name:
    string ( * );
    VAR file_assigned: boolean);
```

3.2.3 SYSTEM UTILITY PROCEDURES

3.2.3.1 Current_Date

The purpose of this procedure is to return the current date in a user selectable format. The procedure reference is on common deck ZPMPDAT and the date formats are on deck ZOSTDAT.

```
{ ZOSTDAT    Returns current date in a user selectable format. }
*calic osddate



*calic osdstat
*calic osddate

{ ZPMPDAT    Contains current date. }

  PROCEDURE [XREF] pmp$get_date ALIAS 'zpmpdat' (format: ost$date_formats;
    VAR date: ost$date;
    VAR status: ost$status);
```

3.2.3.2 Current_Time

The purpose of this procedure is to return the current time of day in a user selectable format. The procedure reference is found on common deck ZPMPTIM and the time formats are found on deck ZOSTTIM.

```
{ ZOSTTIM    Returns current time of day in user selectable format. }
*calic osdtime



*calic osdstat
*calic osdtime

{ ZPMPTIM    Contains current time. }

  PROCEDURE [XREF] pmp$get_time ALIAS 'zpmptim' (format: ost$time_formats;
    VAR time: ost$time;
    VAR status: ost$status);
```

3.2.3.3 Get_NOS_170_Control_Statement_Arguments

The purpose of this procedure is to make available to a CYBIL/CC program, control statements which have been cracked by NOS. Only positional arguments are handled (separators are ignored). The arguments are returned in an adaptable array of seven character strings. The array is both an input and output parameter such that if an actual argument is omitted from the program call, the corresponding element of the array is unaltered. This means the array can be preset with default values for arguments. When more actual arguments are specified on the call than there are elements in the array, the procedure aborts the program with an appropriate dayfile message.

The procedure reference is found on common deck ZUTPCSA.

```
{ ZUTPCSA     Makes continued statements cracked by NOS available to }
{               the calling CYBIL program.                            }

  PROCEDURE [XREF] utp$get_control_statement_args ALIAS 'zutpcsa' (VAR
    args: array [ * ] OF string (7));
```

This interface should only be used when the use of the SES SCL interface (described in System Command Language ERS) is inappropriate. Note that this procedure should be used prior to execution of any OPEN via CYBIO else the first array entry may not contain the contents of the first positional argument.

3.2.3.4 Program's Control Statement as CYBIL String

The purpose of this procedure is to obtain a program's control statement (card) as a CYBIL string. Upon return the control_statement_pointer points to a string of the precise length of the control statement (allocated in the system heap).

Continuation lines are allowed either from a batch job/procedure file stream or from a terminal. Continuation is signaled by terminating the line(s) with an ellipsis (two or more periods). The first character of the continuation line logically replaces the first period of the continuation ellipsis. A control statement of up to 256 characters can be constructed using continuation. When reading continuation lines from an interactive terminal, the prompt:

..?

is issued and should be interpreted as: "enter continuation line". If a control statement longer than 256 characters is entered, the program is aborted by this procedure. The procedure reference is found on common deck ZOSPGCS.

```
*calic osdstat

{ ZOSPGCS    Obtains program's control stmt. (card) as CYBIL string. }

  PROCEDURE [XREF] osp$get_control_statement ALIAS 'zospgcs' (VAR
    control_statement_pointer: ^string ( * );
    VAR status: ost$status);
```

3.2.3.5 Get_Current_User_Name

The purpose of this procedure is to return the current user name. The procedure reference is found on common deck ZUTPGUN.

```
{ ZUTPGUN    Returns the current user name. }

  PROCEDURE [XREF] utp$get_user_name ALIAS 'zutpgun' (VAR user_name: string
    (7);
    VAR user_name_length: 1 .. 7);
```

3.2.3.6 Issue_Dayfile_Message

The purpose of this procedure is to send a message string to the job's dayfile. The string is converted to the 64 character set (6-bit display code) before being sent to the dayfile and line one of the control point. This includes conversion of all lower case letters to upper case. The procedure reference is found on common deck ZUTPMSG. A string of up to 256 characters may be used without fear of truncation.

```
{ ZUTPMSG    Sends message string to job's dayfile. }

  PROCEDURE [XREF] utp$issue_dayfile_message ALIAS 'zutpmsg' (message:
    string ( * ));
```

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.2.3.7 Determine If Job Origin Is Batch

3.2.3.7 Determine_If_Job_Origin_Is_Batch

The purpose of this function is to inform the caller whether or not the executing job was initiated from a batch source. The function examines the NOS 170 job communication area to acquire job origin information. The returned value of the function is set to true or false based on the following job origin types (defined in the NOS V1 Reference Manual Volume 2):

DESCRIPTION	RETURNED VALUE
System	FALSE
Local Batch	TRUE
Remote Batch	TRUE
Time-Sharing	FALSE
Multi-Terminal	FALSE

```
{ ZUTFBOJ    Determine if the job is of batch origin }

  FUNCTION [XREF] utf$batch_origin_job ALIAS 'zutfboj': boolean;
```

Two applications of this function are illustrated below:

```
[ use in IF statements ]

    IF utf$batch_origin_job () THEN
      { do something }
    IFEND;

[ use in assignment statements ]

    var_name := utf$batch_origin_job ();
```

3.2.4 CYBIL PROGRAM PROCEDURES

3.2.4.1 Initiate_CYBIL_Program_Environment

The purpose of this procedure is to initiate the 'environment' for a CYBIL program. The current version just returns the command program name, a pointer to the command line (control statement) that caused the program to be executed, and the command line index ready for scanning the command's parameter list.

```
*calic zostnam
*calic zoststr
*calic osdstat

{ ZOSPINI    Initiates environment for a CYBIL program. }

  PROCEDURE [XREF] osp$initiate ALIAS 'zospini' (VAR command_name:
    ost$name_descriptor;
    VAR command_line_pointer: ^string ( * );
    VAR command_line_index: ost$string_index;
    VAR status: ost$status);
```

The command_line_pointer is obtained using the osp$get_control_statement procedure. See the description of that routine for more information.

3.2.4.2 Terminate a CYBIL Program

The purpose of this procedure is to terminate a CYBIL program. If status.normal is FALSE the message designated by the remaining fields of the status record is issued to the dayfile. If status.normal is FALSE and status.state is osc$error_status or osc$fatal_status the program is aborted, otherwise the program is terminated normally.

```
*calic zostnam
*calic osdstat

{ ZOSPEND    Terminates a CYBIL program. }

  PROCEDURE [XREF] osp$terminate ALIAS 'zospend' (VAR command_name: [READ]
    ost$name_descriptor;
    VAR status: [READ] ost$status);
```

3.2.4.3 Terminate_a_CYBIL_Program_with_Generated_Message

The purpose of this procedure is to terminate a CYBIL program with the option of sending a message to a specified file. The presence of the message generator template array is required. The user has the option to pass a pointer to a CYBIO legible file descriptor, in which case a messages are generated to the file. If no file output is desired the pointer must be set NIL. If status.normal is FALSE the Message Generator is used to generate a message to the dayfile and optionally the specified legible file. If status.normal is FALSE and status.state is osc$error_status or osc$fatal_status the program is aborted, otherwise the program is terminated normally.

```
*CALLC osdstat
*CALLC zostnam

{ZOSPTWM   procedure to terminate CYBIL program with message

PROCEDURE [XREF] osp$terminate_with_message ALIAS 'zosptwm'
    (VAR command_name : [READ] ost$name_descriptor;
     VAR status : [READ] ost$status;
     VAR file_descriptor : file);
```

3.2.4.4 End_CYBIL_Program

The purpose of this procedure is to terminate a CYBIL program. This is accomplished by executing a NOS 170 ENDRUN call.

```
{ ZUTPEND    Terminates a CYBIL program. }

  PROCEDURE [XREF] utp$end ALIAS 'zutpend';
```

3.2.4.5 Abort_CYBIL_Program

The purpose of this procedure is to provide an interface for CYBIL programs to turn off reprieve processing and abort the program.

```
{ ZUTPABT    Aborts a CYBIL program. }

  PROCEDURE [XREF] utp$abort ALIAS 'zutpabt';
```

An additional procedure which can be declared as:

```
  PROCEDURE [XREF] abort;
```

is available also. It differs from the above procedure in that it does not turn off reprieve processing and gives a CYBIL Post Mortem dump.

Another procedure proves useful to suppress terminal output of the last control card on a voluntary abort. A blank message is written to the user dayfile, then a utp$abort is done. The net effect of the blank message is no terminal message.

```
{ZUTPCAA    advance user dayfile with null message and abort(reprieve off)

  PROCEDURE [XREF] utp$clear_and_abort ALIAS 'zutpcaa';
```

3.2.4.6 CYBIL_to_Compass_Interface

The purpose of these macros is to provide a consistent, standard interface from CYBIL to a Compass routine. Three macros are included: ENTR for entry into the compass routine, DONE for exit from the compass routine, and CALL for calling another procedural interface from within a Compass routine.

```
* ZPXIDEF    PROVIDES CYBIL TO COMPASS STANDARD INTERFACE

          CTEXT  ZPXIDEF - CYBIL INTERFACE DEFINITIONS
          SPACE  2
          B1=1
          SPACE  4
*** THE FOLLOWING DEFINES THE NIL POINTER, INDICATING IN CYBIL
*   A POINTER POINTING TO NOTHING


NIL       EQU    3777778
          SPACE  4
*** THE FOLLOWING MACROS DEFINE THE ENTRY/EXIT SEQUENCE OF
*   CYBIL PROCEDURES.

*   ENTRY CONDITIONS
*     B1   1
*     B2   POINTER TO CALLER'S STACK FRAME / TOP OF STACK (TOS)
*     B3   STACK LIMIT
*     X1   1ST ARGUMENT (IF ANY)
*     X2   2ND ARGUMENT (IF ANY)
*     X3   3RD ARGUMENT (IF ANY)
*     X4   4TH ARGUMENT (IF ANY)
*     X5   5TH ARGUMENT (IF ANY)
*     B5   POINTER TO ARGUMENT EXTENSION LIST (IF ANY)
*     X7   PROCEDURE LINKAGE WORD (RETURN ADDRESS)

*   EXIT CONDITIONS
*     B1   1
*     B2   AS ON ENTRY
*     B3   AS ON ENTRY
*     X1   AS X7 ON ENTRY

          SPACE  4
*** THE FOLLOWING MACRO DEFINES THE ENTRY SEQUENCE
*   USING THE CYBIL STACK DISCIPLINE.
```

```
            PURGMAC ENTR

            MACRO   ENTR,NAME
            LOCAL   MORE
MORE        RJ      =XCIL#SPE     * CALL PROLOG EXCEPTION ROUTINE
NAME        SX0     B2            * COPY POINTER TO CALLER'S STACK FRAME
            LX0     18            * POSITION IT
            BX6     X7+X0         * MERGE IT INTO LINKAGE WORD
            SB7     6             * SET ROUTINE STACK FRAME SIZE
            SB2     B2-B7         * ADJUST STACK FRAME POINTER
            GE      B3,B2,MORE    * CHECK IF ROOM IN STACK SEGMENT
            SA6     B2            * STORE LINKAGE WORD INTO STACK
            ENDM
            SPACE   4
*** DONE DEFINES THE CODE SEQUENCE TO RETURN FROM A
*   CYBIL PROCEDURE.

            PURGMAC DONE

DONE        MACRO
            SA1     B2            * LOAD LINKAGE WORD
            SB7     X1            * GET RETURN ADDRESS
            SB2     B2+6          * RESTORE CALLER'S STACK POINTER
            JP      B7            * RETURN
            ENDM
            SPACE   4
*** THE FOLLOWING MACRO DEFINES THE CALLING SEQUENCE FOR A CYBIL
*   PROGRAM.  IT IS ASSUMED, THAT ARGUMENTS ARE ALREADY SET UP.

            PURGMAC CALL

CALL        MACRO   P
            LOCAL   RETAD
            SX7     RETAD         * SET RETURN ADDRESS
            EQ      P             * TRANSFER CONTROL TO PROCEDURE
RETAD       BSS     0
            ENDM
            SPACE   2
            ENDX
```

3.2.5 STRING AND CHARACTER PROCEDURES

3.2.5.1 Compare_CYBIL_Strings

The purpose of this procedure is to compare CYBIL strings which may be of different lengths. The **comparison_result** field contains the result of the comparison.

-1 if left < right string
0 if left = right string
+1 if left > right string.

```
{ ZUTPCPS    Compares CYBIL strings which may be of different lengths. }

  PROCEDURE [XREF] utp$compare_strings ALIAS 'zutpcps' (left_operand:
    string ( * );
    right_operand: string ( * );
    VAR comparison_result: - 1 .. 1);
```

3.2.5.2 Build_Display_Code_String_Pointer

The purpose of this procedure is to build a display code string pointer. Given a cell (word) address and a character position (0 .. 9) within that word receive a display code string pointer. All display code strings are accessed via such pointers.

```
{ ZUTPDCP    Builds a display code string pointer. }

  PROCEDURE [XREF] utp$create_dc_string_ptr ALIAS 'zutpdcp' (word: ^cell;
    pos: 0 .. 9;
    VAR dc_string_ptr: cell);
```

3.2.5.3 Get_Display_Code_Character_From_String

The purpose of this procedure is to get the next display code character from a string. The next display code character designated by the display code pointer is returned and the display code pointer is advanced to designate the following character. Note the following character may be in the next word.

```
{ ZUTPDCG    Gets next display code character from a string. }

  PROCEDURE [XREF] utp$get_next_dc_char ALIAS 'zutpdcg' (VAR dc_string_ptr:
    cell;
    VAR dc_char: 0 .. 3f(16));
```

3.2.5.4 Insert_Display_Code_Character

The purpose of this procedure is to insert a display code character at a place designated by the display code pointer. The display code pointer is advanced to designate the display code character which follows the one inserted. Note that this character may be in the next word.

```
{ ZUTPDCI    Inserts disp. code char. at place designated by pointer. }

  PROCEDURE [XREF] utp$insert_next_dc_char ALIAS 'zutpdci' (VAR
    dc_string_ptr: cell;
    dc_char: 0 .. 3f(16));
```

3.2.6 POINTER MANIPULATION PROCEDURES

3.2.6.1 Offset_of_Pointer_From_Base

This procedure returns the offset (in terms of cells) of an address (pointer) from a base address (pointer).

```
{ ZUTPCOP    Returns offset of address from base address. }

  PROCEDURE [XREF] utp$compute_offset_of_pointer ALIAS 'zutpcop' (base:
    ^cell;
    pointer: ^cell;
    VAR offset: integer);
```

3.2.6.2 Compute_Pointer_From_Offset

This procedure returns a pointer to the "offset-th" cell from a base address (pointer).

```
{ ZUTPCPO    Returns pointer to offset-th cell from base address. }

  PROCEDURE [XREF] utp$compute_pointer_from_offset ALIAS 'zutpcpo' (base:
    ^cell;
    offset: integer;
    VAR pointer: ^cell);
```

3.2.7 CYBIL OVERLAY LOADING

3.2.7.1 Overlay Structures

A parameter for an overlay load is the level numbers. The following structure provides for simple assignment of level numbers prior to the call. It is found on common deck ZUTTOVL.

```
{ZUTTOVL    primary and secondary overlay level numbers

  TYPE
    utt$overlay_level = packed record
      primary: 0 .. 3f(16),
      secondary: 0 .. 3f(16),
    recend;
```

3.2.7.2 Load_Overlay

The following procedure provides the CYBIL user with the capability to load overlays from an overlay file created via LINK170 using an input file created by SES.BOVLAY. Common deck ZUTPOVL provides the procedure interface.

```
*calic zostnam
*calic zuttovl
*calic osdstat

{ZUTPOVL     load an overlay

  PROCEDURE [XREF] utp$load_overlay ALIAS 'zutpovl' (file_name:
    ost$nos170_name;
    level_numbers: utt$overlay_level;
    ptr_to_ovl_proc: ^cell;
    VAR status: ost$status);
```

Before this interface is invoked the environment must be established. An overlay load must be executed from a file. This file must be created by a SES.LINK170 request. The contents of the LGO file given to LINK170 must be of a special form. It must contain specially created compass object routines that externally reference the entry points of procedures/programs which are the outermost of each overlay. It is recommended that the main overlay (0,0) serve as the procedure to contain the overlay loading.

The lower level overlays need no modification except that the outermost procedure must have an XDCLed entry point. The main procedure, which will load the overlays, is expected to use pointers to the procedures which will execute as overlays. When the load of an overlay procedure is executed the #LOC of the pointer to the procedure is passed to the overlay loader. Upon return from the overlay loader the contents of the pointer may be executed as the procedure. Parameters may be passed to the procedure if they were declared as part of the pointer to procedure description. The procedure to be executed as the overlay is never directly referenced in the procedure loading the overlay. Procedures to be used as overlays to a procedure may be XREFerenced in that procedure, but it cannot be performed by name.

Any variables shared by overlayed procedures may be accomodated by XDCL,XREF. Lower level overlays may not have more than one entry point.

Below is an example of what is required to alter the CYBIL source code to use overlays.

The following is a sketch of some basic given elements of an existing program.

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.2.7.2 Load Overlay

```
          .
          .
          .
procedure [XREF] utp$initiate (VAR status : ost$status);
          .
          .
          .
utp$initiate (status);
          .
          .
          .
```

The following is a sketch of an overlay load:

```
VAR
  ptr_to_utp$initiate : ^procedure (VAR status: ost$status);

    .
    .
  overlay_level.primary := 1;
  overlay_level.secondary := 2;

  utp$load_overlay (overlay_file, overlay_level,
    #LOC(ptr_to_utp$initiate), status);
      .
      .
  ptr_to_utp$initiate^ (status);
      .
      .
```

The assumptions are:

(1) the procedure pointed to by ptr_to_utp$initiate is a procedure XDCLed and having a single parameter status,

(2) the overlay file name is the name of the overlay file (b parameter) created by SES.LINK170 using a lgo created by SES.BOVLAY.

The overlayed procedure may be XREFerenced in the procedure calling it but referencing it by name will cause it to link there.

Note the convention used to label the pointer to procedure variable. It is suggested that overlay procedures use a common deck of the following format:

```
PROCEDURE [XREF] utp$initiate alias 'zutpini' (VAR status : ost$status);
```

```
VAR
  ptr_to_utp$initiate: ^procedure (VAR status: ost$status);
```

This allows a program library cross reference to locate the place of use of a procedure. XREFed CYBIL procedures do not link unless they are performed by name in the compiled code.

3.2.7.3 Create Overlay Tree Structure (SES.BOVLAY)

This procedure is intended to create a LGO file consisting of overlay cards and relocatable binaries that have external references to entry points of the overlay procedures. This file is input to SES.LINK170 to create an executable overlay file.

SES.BOVLAY i = l = b =

i	(optional) the name of input file containing data used to create compass source code decks (default is INPUT).
l	(optional) name of file to contain the list of assembled code composing generated overlay linkage decks. Default is LIST.
b	(optional) name of a load and go format file created by assembling the generated compass source decks (default is LGO). It is intended that this LGO file is used as input (f parameter) to SES.LINK170 to create an overlay file.

The input data to SES.BOVLAY is of the following form:

ext	(required) 7 character name of entry point (alias) of procedure (program if level 0,0) which was XDCLed and is the outermost of the overlay.
ovl	(required) the decimal level number of the overlay associated with the entry point (p,s) where p is the primary level and s is secondary level.
ent	(optional) 1..7 character name of entry point (which is the transfer address) for the deck created. It may prove useful to indicate the level number as part of the name. If not specified a unique name is generated.
ident	(optional) name of ident card of compass routine created. It may prove useful to use a symbol which

indicates the overlay level number. If not specified a unique name is generated.

Note that the input data must be ordered in overlayed sequence-- that is (0,0) (1,0) (1,1) (2,0) etc. Up to 77(8) levels are allowed. See the CYBER Loader Reference Manual for further details on overlaying.

When input is not specified a prompt is issued giving a suggested data ordering. Then data is input without keyword assignment ('keyword =...') it must be in the suggested order. When assignment is used the parameters may be scrambled on the input line. Blanks or commas may be used as seperators. If a parameter is ommitted prior to one shich is specified, commas are required to note the absence. An example of input data follows:

```
zutpini 1 2 zuteo12 zutio12
zutpout 1 3, , zutio13
zutpexp 1 4
zutpmin 1 5 zuteo15
```

The binary decks are created from generated source code:

```
        IDENT   ident
        ENTRY   ent
        EXT     ext
        LCC     OVERLAY(,p,s)
ent     EQ      ext
        END     ent
```

The deck simply causes generation of an overlay card and a binary which references the entry point of the CYBIL procedure which is to become an overlay.

3.2.8 USER DETECTION OF TERMINAL INTERRUPT CONDITION

User condition processing is a limited set based upon the existence of extended reprieve capabilities within CYBIL runtime. This also depends on a NOS 170 system at or later than R4 level.

3.2.8.1 Interruptable Condition Request Codes

The request codes correspond to the condition mask bits of the NOS 170 REPRIEVE/RECOVR. Note that a limitation stated in the NOS 170 reference manual excludes terminal interrupts, but since CYBIL CC has pseudo RECOVR code the terminal interrupt is allowed. The common deck is ZUTTRCV.

```
{ ZUTTRCV    RECOVR mask conditions

  TYPE
    utt$request_codes = (terminal_interrupt, normal_end, cp_abort,
      pp_abort, operator_action, limits_exceeded, pp_error, mode_error),
      { 200, 100, 040, 020, 010, 004, 002, 001 }

    utt$recovr_request = set of utt$request_codes;
```

3.2.8.2 Initialize_Terminal_Interrupt_Detection

This procedure establishes a user recovery routine to detect terminal interrupt conditions. This is accomplished by using **utp$activate_recovr_request** with a mask only for terminal interrupt condition and a pointer to procedure **utp$record_terminal_interrupt**. An externally declared variable **utv$terminal_interrupt_count** is incremented each time the procedure is invoked. After a count of three is reached the program is aborted. It is expected that the user will periodically use **utp$terminal_interrupt_detected** to examine the count and clear it.

```
{ZUTPITI    initialize terminal interrupt detection

  PROCEDURE [XREF] utp$init_term_interrupt_detect ALIAS 'zutpiti';
```

3.2.8.3 Was_Terminal_Interrupt_Detected

This function detects the occurrence of a terminal interrupt at a user selected time during the execution of a user program. It is intended to be used at a time convenient to the user. The variable **utv$terminal_interrupt_count** is checked for a non zero value and reset to zero. The boolean value is set true when the count was non zero else false. The user is free to direct action depending upon his needs. Checking for past occurence of a terminal interrupt avoids the immediate danger of attempting to invoke a user process that may use non-reentrant code of the NOS 170 system.

```
[ZUTPTID    (function)latent check for terminal interrupt

  FUNCTION [XREF] utf$terminal_interrupt_detected ALIAS 'zutftid'
    : boolean;
```

3.2.8.4 Ask_for_Direction

This procedure is supplied for the user who wishes to ask the question 'QUIT OR RESUME' upon detecting a terminal interrupt. A response of "RESUME" sets end_operation equal false while "QUIT" sets end_operation to "true". This routine has a dependency on the message template array of the message generator and requires use of product code 'UT' when the SES.GENMAR proc is used to create the template array.

```
*calic zn7txch

{ZUTPASK    procedure to prompt user for direction when terminal
{interrupt recognized

  PROCEDURE [XREF] utp$ask_for_direction ALIAS 'zutpask' (VAR
    end_operation: boolean);
```

3.2.9 GENERAL PROCEDURES

3.2.9.1 Generate_Unique_Alphanumeric_Strings

There are three procedures available to generate unique alphanumeric character strings. These procedures generate unique strings, labels and file names. There are three separate procedure reference common decks.

```
{ ZUTPUQS    Generates unique string. }

  PROCEDURE [XREF] utp$generate_unique_string ALIAS 'zutpuqs' (VAR
    unique_string: string ( * ));
```

```
{ ZUTPUQL    Generates unique label. }

  PROCEDURE [XREF] utp$generate_unique_label ALIAS 'zutpuql' (VAR
    unique_label: string (7));
```

Note that all labels produced are of the form 9Qxxxxx where xxxxx are the unique charcters generated.

```
{ ZUTPUQF    Generates unique file name. }

  PROCEDURE [XREF] utp$generate_unique_file_name ALIAS 'zutpuqf' (VAR
    unique_file_name: string (7));
```

The file names are of the form ZQxxxxx where xxxxx are the unique characters generated. Note that the generated file name is guaranteed to be different from the name of any file currently assigned to the job.

3.2.10 CYBIL TO NOS 170 PROCEDURES

A common deck which is needed in many of the NOS 170 interface procedures is ZN7TJCA. It contains the format of the Job Communication Area.

```
{ ZN7TJCA    Format of Job Communication Area for CYBIL to NOS 170. }

?? fmt ( format := off ) ??

  TYPE
    n7t$job_communication_area = PACKED RECORD
      res1     : SET OF 1 .. 45,                          { RA + 0 }
      cf       : BOOLEAN,                 { CFO bit }
      res2     : SET OF 1 .. 1,
      p        : BOOLEAN,                 { pause flag }
                                          { sense switches }
      ssw      : PACKED ARRAY[ - 6 .. - 1] OF BOOLEAN,
                                          { FORTRAN switches }
      fsw      : PACKED ARRAY[ - 6 .. - 1] OF BOOLEAN,
                                                     { RA + 1 }
      sname    : 0 .. 3FFFF(16),          { sys req name }
      unused1  : SET OF 1 .. 1,
      r        : BOOLEAN,                 { auto recall flag }
      unused2  : SET OF 1 .. 4,
      sargs    : 0 .. 0FFFFFFFFF(16),     { sys req args }
                                                     { RA + 2..63 }
      argr     : ARRAY[1 .. 32(16)] OF PACKED RECORD
        arg    : 0 .. 3FFFFFFFFF(16),     { parmater }
        sep    : 0 .. 3FFFF(16),          { separator }
      RECEND,
                                                     { RA + 64 }
      pgnr     : 0 .. 3FFFFFFFFF(16),     { prog name }
      actr     : 0 .. 3FFFF(16),          { argument count }
                                                     { RA + 65 }
      cmur     : BOOLEAN,                 { CMU flag }
      unused3  : SET OF 1 .. 40,
      lwpr     : BOOLEAN,                 { loader flag }
      nwai     : ^CELL,                   { next word avail for load }
                                                     { RA + 66 }
      xjpr     : BOOLEAN,                 { CEJ/MEJ flag }
      cpu0_is  : BOOLEAN,                 { CPU0 has inst stack }
      cpu1     : BOOLEAN,                 { CPU1 is present }
      res3     : SET OF 1 .. 4,
```

```
      pp_#     : 0 .. 1F(16),              { number of PPs }
      cm_size  : 0 .. 0FFF(16),            { CM size }
      jopr     : 0 .. 0FFF(16),            { job origin }
      unused4  : SET OF 1 .. 4,
      dis      : BOOLEAN,                  { DIS flag }
      rss      : BOOLEAN,                  { RSS flag }
      fwpr     : ^CELL,                    { first word of prog }
                                                        { RA + 67 }
      csmr     : BOOLEAN,                  { char set mode }
      unused5  : SET OF 1 .. 29,
      idrr     : BOOLEAN,                  { loader completion flag }
      unused6  : SET OF 1 .. 29,
                                                        { RA + 70 }
      ccdr     : ALIGNED[0 MOD 8] ARRAY[1 .. 8] OF
                          PACKED ARRAY[0 .. 9] OF 0 .. 3F(16),
    RECEND;                                             { RA + 100 }
```

```
?? fmt ( format := on ) ??
```

This can be made available as a variable in the program area with the following declaration:

```
  VAR
    jca ALIAS *SW=RA0* : [XREF] n7t$job_communication_area;
```

3.2.10.1 NOS_170_Combined_Input_Output_(CIO)_Request

There are two procedures to interface CYBIL with NOS 170 CIO. The only difference between the two is the skip operations parameter. They are available on common deck ZN7PCIO.

This interface is intended primarily for use by "higher-level" utility routines.

```
*calic zn7tfet

{ ZN7PCIO    NOS 170 combined input output (CIO) request. }

  CONST { CIO request codes }
    n7c$cio_rphr = 0(16),
    n7c$cio_read = 8(16),
    n7c$cio_readskp = 10(16),
    n7c$cio_readcw = 80(16),
    n7c$cio_readls = 88(16),
    n7c$cio_rphrls = 98(16),
    n7c$cio_readns = 0a8(16),
    n7c$cio_readei = 180(16),
    n7c$cio_wphr = 4(16),
    n7c$cio_write = 0c(16),
    n7c$cio_writer = 14(16),
    n7c$cio_writef = 1c(16),
    n7c$cio_writecw = 84(16),
    n7c$cio_rewrite = 8c(16),
    n7c$cio_rewriter = 94(16),
    n7c$cio_rewritef = 9c(16),
    n7c$cio_open_read_norewind = 40(16),
    n7c$cio_open_read_rewind = 60(16),
    n7c$cio_open_write_norewind = 44(16),
    n7c$cio_open_write_rewind = 64(16),
    n7c$cio_open_alter_norewind = 50(16),
    n7c$cio_open_alter_rewind = 70(16),
    n7c$cio_close_norewind = 58(16),
    n7c$cio_close_rewind = 68(16),
    n7c$cio_close_unload = 78(16),
    n7c$cio_close_return = 7c(16),
    n7c$cio_bksp = 20(16),
    n7c$cio_bkspru = 24(16),
    n7c$cio_rewind = 28(16),
    n7c$cio_unload = 30(16),
    n7c$cio_return = 38(16),
    n7c$cio_evict = 4c(16),
```

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.2.10.1 NOS 170 Combined Input Output (CIO) Request

```
    n7c$cio_skipr = 0a0(16),
    n7c$cio_skipf = 3c0a0(16),
    n7c$cio_skiprb = 1a0(16),
    n7c$cio_skipfb = 3c1a0(16),
    n7c$cio_skipei = n7c$cio_skipf,
    n7c$cio_eoi_skip_count = 3ffff(16);

  PROCEDURE [XREF] n7p$cio_with_skip ALIAS 'zn7pios' (VAR fet: n7t$fet;
    request_code: - 3ffff(16) .. 3ffff(16);
    skip_count: 0 .. n7c$cio_eoi_skip_count);

  PROCEDURE [XREF] n7p$cio ALIAS 'zn7pcio' (VAR fet: n7t$fet;
    request_code: - 3ffff(16) .. 3ffff(16));
```

For an explanation of the CIO functions consult the NOS 170 Reference Manual Volume 2.

3.2.10.2 NOS_170_Control_Point_Manager_(CPM)

There are two procedures to allow the CYBIL user to alter or interrogate parameters in the job control point area which controls his job in the system. The choice of procedure is dependent on the request code. The subfunction_code is usually set to zero. The procedure references and some constant request codes are available on common deck ZN7PCPM.

This interface is intended primarily for use by "higher-level" utility routines.

```
{ ZN7PCPM    NOS 170 Control Point Manager (CPM). }

  CONST { CPM request codes }
    n7c$cpm_setqp = 0(16),
    n7c$cpm_setpr = 1(16),
    n7c$cpm_mode = 2(16),
    n7c$cpm_setasl = 3(16),
    n7c$cpm_setjsl = 3(16),
    n7c$cpm_settl = 3(16),
    n7c$cpm_erexit = 4(16),
    n7c$cpm_console = 5(16),
    n7c$cpm_rollout = 6(16),
    n7c$cpm_getasl = 7(16),
    n7c$cpm_jsl = 7(16),
    n7c$cpm_setssm = 8(16),
    n7c$cpm_onsw = 9(16),
    n7c$cpm_offsw = 0a(16),
    n7c$cpm_getjn = 0b(16),
    n7c$cpm_getqp = 0c(16),
    n7c$cpm_getpr = 0d(16),
    n7c$cpm_getem = 0e(16),
    n7c$cpm_gettl = 0f(16),
    n7c$cpm_setdfri = 10(16),
    n7c$cpm_setui = 11(16),
    n7c$cpm_setlc = 12(16),
    n7c$cpm_setrfl = 13(16),
    n7c$cpm_getjcr = 14(16),
    n7c$cpm_setjcr = 15(16),
    n7c$cpm_setss = 16(16),
    n7c$cpm_getjo = 17(16),
    n7c$cpm_getja = 18(16),
    n7c$cpm_usecpu = 19(16),
    n7c$cpm_usernum = 1a(16),
```

```
    n7c$cpm_getflc = 1b(16),
    n7c$cpm_setpacknam = 1d(16),
    n7c$cpm_getpacknam = 1e(16),
    n7c$cpm_getss = 1f(16),
    n7c$cpm_version = 24(16),
    n7c$cpm_getlc = 25(16),
    n7c$cpm_getgls = 26(16),
    n7c$cpm_setgls = 27(16),
    n7c$cpm_machid = 28(16),
    n7c$cpm_getact = 29(16),
    n7c$cpm_setmfl = 2a(16),
    n7c$cpm_getpfp = 2f(16),
    n7c$cpm_getlof = 31(16),
    n7c$cpm_setlof = 32(16),
    n7c$cpm_getjci = 3c(16),
    n7c$cpm_setjci = 3c(16),
    n7c$cpm_protect = 3d(16);

  PROCEDURE [XREF] n7p$cpm_with_value ALIAS 'zn7pcpm' (value: -
    1ffff(16) .. 1ffff(16);
    request_code: n7c$cpm_setqp .. n7c$cpm_protect;
    subfunction_code: 0 .. 3f(16));

  PROCEDURE [XREF] n7p$cpm_with_pointer ALIAS 'zn7pcpm' (pointer: ^cell;
    request_code: n7c$cpm_erexit .. n7c$cpm_protect;
    subfunction_code: 0 .. 3f(16));
```

For an explanation of the CPM functions consult the NOS 170 Reference Manual Volume 2.

The following decks describe the format of information for use in making CPM requests.

```
{ ZN7TEMR    Contains type definition for exit mode. }

  TYPE
    n7t$exit_mode = packed record
      fill: set of 1 .. 48,
      em: 0 .. 0fff(16),
    recend;
```

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.2.10.2 NOS 170 Control Point Manager (CPM)

```
{ ZN7TFLR    Contains type definition for field length. }

  TYPE
    n7t$field_length = packed record
      jmfl: 0 .. 0fff(16),
      icfl: 0 .. 0fff(16),
      fill: 0 .. 0fff(16),
      rifl: 0 .. 0fff(16),
      flir: 0 .. 0fff(16),
    recend;



{ ZN7TJCR    Contains type definition for job control registers. }

  TYPE
    n7t$job_control_registers = packed record
      ef: 0 .. 3f(16),
      r3: - 1ffff(16) .. 1ffff(16),
      r2: - 1ffff(16) .. 1ffff(16),
      r1: - 1ffff(16) .. 1ffff(16),
    recend;



{ ZN7TRCR    Contains type definition for rollout control. }

  TYPE
    n7t$rollout_control = packed record
      fill: set of 1 .. 30,
      evd: 0 .. 3ffff(16),
      rtp: 0 .. 0fff(16),
    recend;



{ ZN7TSET    Contains NOS 170 symbols for error types. }

  CONST
    n7c$aret = 1(16),
    n7c$pset = 2(16),
    n7c$ppet = 3(16),
```

```
    n7c$cpet = 4(16),
    n7c$pcet = 5(16),
    n7c$tlet = 6(16),
    n7c$flet = 7(16),
    n7c$tket = 8(16),
    n7c$sret = 9(16),
    n7c$fset = 0a(16),
    n7c$odet = 0b(16),
    n7c$spet = 0c(16),
    n7c$rret = 0c(16),
    n7c$oket = 0d(16),
    n7c$sset = 0e(16),
    n7c$ecet = 0f(16),
    n7c$peet = 10(16),
    n7c$syet = 11(16),
    n7c$oret = 12(16);



{ ZN7TSJO    NOS 170 symbols for job origin type. }

  CONST { NOS 170 symbols for job origin types }
    n7c$syot = 0,
    n7c$bcot = 1,
    n7c$eiot = 2,
    n7c$txot = 3;



{ ZN7TSSS    NOS 170 symbols for subsystems. }

  CONST { NOS 170 symbols for sub-systems }
    n7c$nuls = 0,
    n7c$bass = 1,
    n7c$fors = 2,
    n7c$ftns = 3,
    n7c$exes = 4,
    n7c$bats = 5,
    n7c$accs = 6,
    n7c$tras = 7;
```

3.2.10.3 NOS_170_Local_File_Manager_(LFM)

This procedure allows the CYBIL user to interface to the NOS 170 Local File Manager. The **setid_code** parameter should be zero except for a request of **n7c$lfm_setid**. The procedure reference, function constants and error code constants are on common deck ZN7PLFM.

This interface is intended primarily for use by "higher-level" utility routines. It may be necessary to set other fields in the fet in order to meet request requirements. Consult the NOS Reference Manual Vol 2 for details.

```
*calic zn7tfet

{ ZN7PLFM    Allows CYBIL user interface to NOS 170 Local File Manager. }

  CONST { LFM request codes }
    n7c$lfm_rename = 0(16),
    n7c$lfm_assign01 = 1(16),
    n7c$lfm_common = 2(16),
    n7c$lfm_release03 = 3(16),
    n7c$lfm_print = 4(16),
    n7c$lfm_punch = 5(16),
    n7c$lfm_punchb = 6(16),
    n7c$lfm_p8 = 7(16),
    n7c$lfm_lock = 8(16),
    n7c$lfm_unlock = 9(16),
    n7c$lfm_status12 = 0a(16),
    n7c$lfm_status13 = 0b(16),
    n7c$lfm_request14 = 0c(16),
    n7c$lfm_request15 = 0d(16),
    n7c$lfm_batch = 0e(16),
    n7c$lfm_setid = 0f(16),
    n7c$lfm_assign20 = 10(16),
    n7c$lfm_accsf = 11(16),
    n7c$lfm_encsf = 12(16),
    n7c$lfm_pscsf = 13(16),
    n7c$lfm_label = 14(16),
    n7c$lfm_getfnt = 15(16),
    n7c$lfm_request26 = 16(16),
    n7c$lfm_entervsn = 17(16),
    n7c$lfm_release30 = 18(16),
    n7c$lfm_primary = 19(16),
    n7c$lfm_filinfo = 20(16);

  CONST { LFM error codes }
```

```
    n7c$lfm_ok = 0(16),
    n7c$lfm_file_found = 0(16),
    n7c$lfm_file_not_found = 1(16),
    n7c$lfm_file_name_error = 2(16),
    n7c$lfm_illegal_file_type = 3(16),
    n7c$lfm_file_empty = 4(16),
    n7c$lfm_magnet_not_active = 5(16),
    n7c$lfm_duplicate_lib_file_name = 6(16),
    n7c$lfm_illegal_equipment = 7(16),
    n7c$lfm_equipment_not_available = 8(16),
    n7c$lfm_duplicate_file_name = 9(16),
    n7c$lfm_illegal_user_access = 0a(16),
    n7c$lfm_illegal_user_number = 0b(16),
    n7c$lfm_illegal_id_code = 0c(16),
    n7c$lfm_resex_detected_error = 0d(16),
    n7c$lfm_io_sequence_error = 0e(16),
    n7c$lfm_output_file_limit = 0f(16),
    n7c$lfm_local_file_limit = 10(16),
    n7c$lfm_no_mass_storage = 11(16),
    n7c$lfm_illegal_file_mode = 12(16),
    n7c$lfm_fet_too_short = 13(16),
    n7c$lfm_getfnt_table_too_large = 14(16),
    n7c$lfm_bad_change_file_org_typ = 15(16),
    n7c$lfm_parameter_block_busy = 16(16),
    n7c$lfm_address_out_of_range = 17(16);

  TYPE
    n7t$lfm_error_codes = 0 .. 0ff(16);

  PROCEDURE [XREF] n7p$lfm ALIAS 'zn7plfm' (request_code: n7c$lfm_rename ..
    n7c$lfm_filinfo;
    VAR fet: n7t$fet;
    setid_code: 0 .. 3f(16));
```

For an explanation of the LFM function and error codes consult the NOS 170 Reference Manual Volume 2.
The ZN7TSFT common deck contains the file type constants.

```
{ ZN7TSFT    Contains NOS 170 symbols for file types. }

  CONST { NOS 170 symbols for file types }
    n7c$inft = 0(16),
    n7c$roft = 1(16),
    n7c$prft = 2(16),
```

```
      n7c$phft = 3(16),
      n7c$teft = 4(16),
      n7c$quft = 5(16),
      n7c$syft = 5(16),
      n7c$loft = 6(16),
      n7c$cmft = 7(16),
      n7c$lift = 8(16),
      n7c$ptft = 9(16),
      n7c$pmft = 0a(16),
      n7c$faft = 0b(16),
      n7c$hsft = 0c(16),
      n7c$lcft = 0d(16),
      n7c$cnft = 0e(16),
      n7c$mxft = 0f(16);
```

3.2.10.4 Message_to_NOS_170_Dayfile

This procedure allows the CYBIL user to issue a display code message to the dayfile. Note that the message is also sent to line one of the control point. The procedure reference is on common deck ZN7PMSG.

```
{ ZN7PMSG    Allows CYBIL user to issue disp. code msg. to NOS 170
{dayfile. }

  PROCEDURE [XREF] n7p$issue_dayfile_message ALIAS 'zn7pmsg'
    (ptr_to_dc_message: ^cell;
    destination_code: 0 .. 7);
```

For a complete explanation of the options for destination code consult the MESSAGE description in the NOS 170 Reference Manual Volume 2.

This interface is intended primarily for "higher-level" utility routines. Another routine (utp$issue_dayfile_message) is available that accepts a CYBIL string as the message text, and is therefore generally more useful for most applications.

3.2.10.5 NOS_170_Permanent_File_Manager_(PFM)

This procedure allows the CYBIL user to interface to the NOS 170 Permanent File Manager. The procedure reference, request codes, category codes, access mode codes and error codes are on common deck ZN7PPFM.

This interface is intended primarily for use by "higher-level" utility routines.

```
*calic zn7tfet

{ ZN7PPFM    Allows CYBIL user interface to NOS 170 PFM. }

  CONST { PFM request codes }
    n7c$pfm_save = 1,
    n7c$pfm_get = 2,
    n7c$pfm_purge = 3,
    n7c$pfm_catlist = 4,
    n7c$pfm_permit = 5,
    n7c$pfm_replace = 6,
    n7c$pfm_append = 7,
    n7c$pfm_define = 8,
    n7c$pfm_attach = 9,
    n7c$pfm_change = 10;

  CONST { PFM file category codes }
    n7c$pfm_ct_private = 0,
    n7c$pfm_ct_semi_private = 1,
    n7c$pfm_ct_public = 2;

  CONST { PFM file access mode codes }
    n7c$pfm_m_write = 0,
    n7c$pfm_m_read = 1,
    n7c$pfm_m_append = 2,
    n7c$pfm_m_execute = 3,
    n7c$pfm_m_null = 4,
    n7c$pfm_m_modify = 5,
    n7c$pfm_m_read_modify = 6,
    n7c$pfm_m_read_append = 7;

  TYPE
    n7t$pfm_modes = n7c$pfm_m_write .. n7c$pfm_m_read_append;

  CONST { PFM error codes }
    n7c$pfm_ok = 0(16),
    n7c$pfm_file_found = 0(16),
```

```
      n7c$pfm_file_busy = 1(16),
      n7c$pfm_file_not_found = 2(16),
      n7c$pfm_file_empty = 3(16),
      n7c$pfm_file_not_on_mass_storag = 4(16),
      n7c$pfm_file_already_permanent = 5(16),
      n7c$pfm_file_not_local = 6(16),
      n7c$pfm_file_name_error = 7(16),
      n7c$pfm_illegal_user_access = 8(16),
      n7c$pfm_illegal_device_request = 9(16),
      n7c$pfm_file_too_long = 0a(16),
      n7c$pfm_illegal_request = 0b(16),
      n7c$pfm_device_unavailable = 0c(16),
      n7c$pfm_illegal_file_type = 0d(16),
      n7c$pfm_pf_utility_active = 0e(16),
      n7c$pfm_data_transfer_error = 0f(16),
      n7c$pfm_catalog_overflow_files = 10(16),
      n7c$pfm_catalog_overflow_size = 11(16),
      n7c$pfm_prus_not_available = 12(16),
      n7c$pfm_io_sequence_error = 13(16),
      n7c$pfm_local_file_limit = 14(16),
      n7c$pfm_pru_limit = 15(16),
      n7c$pfm_permit_limit_exceeded = 16(16),
      n7c$pfm_reserved_27 = 17(16),
      n7c$pfm_sys_resex_failure_30 = 18(16),
      n7c$pfm_sys_track_limit = 19(16),
      n7c$pfm_sys_file_length_error = 1a(16),
      n7c$pfm_sys_random_index_error = 1b(16),
      n7c$pfm_sys_dir_acc_file_error = 1c(16),
      n7c$pfm_sys_replace_error = 1d(16),
      n7c$pfm_sys_pfm_abort = 1e(16),
      n7c$pfm_sys_mass_storage_error = 1f(16),
      n7c$pfm_sys_file_data_error = 20(16),
      n7c$pfm_sys_permit_error = 21(16),
      n7c$pfm_sys_data_permit_error = 22(16),
      n7c$pfm_sys_eoi_changed = 23(16),
      n7c$pfm_sys_resex_failure_44 = 24(16),
      n7c$pfm_reserved_45 = 25(16),
      n7c$pfm_reserved_46 = 26(16),
      n7c$pfm_reserved_47 = 27(16),
      n7c$pfm_sys_file_structure_err = 28(16),
      n7c$pfm_sys_system_sector_error = 29(16),
      n7c$pfm_reserved_52 = 2a(16),
      n7c$pfm_reserved_53 = 2b(16),
      n7c$pfm_reserved_54 = 2c(16),
      n7c$pfm_reserved_55 = 2d(16),
      n7c$pfm_reserved_56 = 2e(16),
      n7c$pfm_reserved_57 = 2f(16),
      n7c$pfm_reserved_60 = 30(16),
```

```
    n7c$pfm_reserved_61 = 31(16),
    n7c$pfm_reserved_62 = 32(16),
    n7c$pfm_reserved_63 = 33(16),
    n7c$pfm_reserved_64 = 34(16),
    n7c$pfm_reserved_65 = 35(16),
    n7c$pfm_reserved_66 = 36(16),
    n7c$pfm_reserved_67 = 37(16),
    n7c$pfm_reserved_70 = 38(16),
    n7c$pfm_sys_staging_error = 39(16),
    n7c$pfm_file_being_staged = 3a(16),
    n7c$pfm_file_awaiting_staging = 3b(16),
    n7c$pfm_file_not_available = 3c(16),
    n7c$pfm_file_is_direct = 3d(16),
    n7c$pfm_file_is_indirect = 3e(16),
    n7c$pfm_reserved_77 = 3f(16),
    n7c$pfm_file_stagable = 40(16),
    n7c$pfm_sys_pfc_address_error = 41(16),
    n7c$pfm_sys_pfc_data_error = 42(16),
    n7c$pfm_non_stagable_request = 43(16),
    n7c$pfm_interlock_not_available = 44(16),
    n7c$pfm_alt_image_obsolete = 45(16),
    n7c$pfm_sys_alt_storage_error = 46(16),
    n7c$pfm_fnt_full = 47(16),
    n7c$pfm_alt_image_not_obsolete = 48(16),
    n7c$pfm_activity_count_limit = 49(16);

  TYPE
    n7t$pfm_error_codes = 0 .. 0ff(16);

  PROCEDURE [XREF] n7p$pfm ALIAS 'zn7ppfm' (request_code: n7c$pfm_save ..
    n7c$pfm_change;
    VAR fet: n7t$fet);
```

For further details on the request codes available consult the NOS 170 Reference Manual Volume 2.

3.2.10.6 NOS_170_Recall

This procedure allows the CYBIL user to interface to the NOS 170 system RECALL facility. It enables the CYBIL user to relinquish the CPU until the recall time has elapsed. The procedure reference is available on common deck ZN7PRCL.

```
{ ZN7PRCL    Allows CYBIL user interface to NOS 170 RECALL facility. }

  PROCEDURE [XREF] n7p$recall ALIAS 'zn7prcl';
```

For a detailed explanation of RECALL consult the NOS 170 Reference Manual Volume 2.

3.2.10.7 NOS_170_Translate_Control_Statement

This procedure allows the CYBIL user to interface to the NOS 170 translate control statement facility. A user may read a control statement from or place a control statement in the control statement stream. The procedure reference, request codes and sub-function codes are available on common deck ZN7PTCS.

This interface is intended primarily for use by "higher-level" utility routines.

```
{ ZN7PTCS    Allows CYBIL user interface to NOS 170 trans. cont. stmt. }

  CONST { TCS request codes }
    n7c$tcs_read = 4,
    n7c$tcs_execute = 5;

  CONST { TCS sub-function codes }
    n7c$tcs_read_and_advance = 0,
    n7c$tcs_read_if_not_local_file = 1,
    n7c$tcs_read_even_if_local_file = 2,
    n7c$tcs_add_for_nosbe_format = 4;

  PROCEDURE [XREF] n7p$translate_control_statement ALIAS 'zn7ptcs'
    (request_code: n7c$tcs_read .. n7c$tcs_execute;
    sub_function: 0 .. 6;
    ptr_to_dc_control_statement: ^cell);
```

A detailed explanation of the request codes and sub-function may be found in the NOS 170 Reference Manual Volume 2.

3.2.10.8 NOS_170_Time_Processor

There are a variety of requests available to the CYBIL user through the n7p$time interface. Most of the requests are processed by the system monitor directly rather than through a specific function processor. There are requests to return the current time of day in display code, return the current date in display code, return the current Julian date, return the current date and time in packed format, return the real elapsed time since deadstart, return the accumulated system resource units, return the accumulated central processor time used by the job, and return the packed time as display code. The request codes and procedure reference are available on common deck ZN7PTIM.

This interface is intended primarily for use by "higher-level" utility routines. Other routines (pmpget_date, pmpget_time) exist to perform the more common requests for most applications.

```
{ ZN7PTIM    Contains NOS 170 time processor information. }

  CONST { TIM request codes }
    n7c$tim_jobs_cpu_time = 0,
    n7c$tim_date = 1,
    n7c$tim_clock = 2,
    n7c$tim_julian_date = 3,
    n7c$tim_jobs_real_time = 4,
    n7c$tim_time_since_deadstart = 5,
    n7c$tim_packed_date_and_clock = 6,
    n7c$tim_jobs_srus = 7;

  PROCEDURE [XREF] n7p$time ALIAS 'zn7ptim' (request_code:
    n7c$tim_jobs_cpu_time .. n7c$tim_jobs_srus;
    ptr_to_response_word: ^cell);
```

The details concerning the request codes may be found in the NOS 170 Reference Manual Volume 2 in the System Requests chapter.

The following decks describe the format of returned information.

```
{ ZN7TJCT    Contains type definition of job's cpu time. }

  TYPE
    n7t$jobs_cpu_time = packed record
```

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.2.10.8 NOS 170 Time Processor

```
      fill: set of 1 .. 3,
      ss: 0 .. 1fffffffffff(16),
      ms: 0 .. 0fff(16),
    recend;



{ ZN7TJDR    Contains type definition of julian date. }

  TYPE
    n7t$julian_date = packed record
      fill: set of 1 .. 30,
      jd: packed array[1 .. 5] of 0 .. 3f(16),
    recend;



{ ZN7TJRT    Contains type definition of job's real time. }

  TYPE
    n7t$jobs_real_time = packed record
      fill: set of 1 .. 24,
      seconds_times_4096: 0 .. 0fffffffff(16),
    recend;



{ ZN7TJST    Contains type definition of job's system time. }

  TYPE
    n7t$jobs_system_time = packed record
      fill: set of 1 .. 24,
      srus: 0 .. 0fffffffff(16),
    recend;



{ ZN7TPDC    Contains type definition of date and time. }

  TYPE
```

```
    n7t$date_clock = packed record
      fill: set of 1 .. 24,
      year_minus_1970: 0 .. 3f(16),
      month: 0 .. 3f(16),
      day: 0 .. 3f(16),
      hour: 0 .. 3f(16),
      minute: 0 .. 3f(16),
      second: 0 .. 3f(16),
    recend;
```

```
{ ZN7TTSD    Contains type definition of time since deadstart. }

  TYPE
    n7t$time_since_deadstart = packed record
      ss: 0 .. 0ffffff(16),
      ms: 0 .. 0fffffffff(16),
    recend;
```

3.2.10.9 NOS_170_Wait_Not_Busy

This procedure provides a mechanism for the CYBIL user to interface to the NOS 170 wait not busy process. This allows a user to wait for completion of an I/O operation. The status word is the word 0 bit 0 of the FET. It should be pointed out that waiting for an I/O operation is the most common usage but the status word could be used in other operations such as a memory request. In this case the status word may be any word in program central memory. The procedure reference is available on common deck ZN7PWNB.

```
{ ZN7PWNB    Allows CYBIL user interface to NOS 170 wait not busy. }

  PROCEDURE [XREF] n7p$wait_not_busy ALIAS 'zn7pwnb' (ptr_to_status_word:
    ^cell);
```

This amounts to a RECALL with a status word specified. For further detail see the RECALL explanation in the NOS 170 Reference Manual Volume 2.

3.2.11 CHECKPOINT FILE ROUTINES

The checkpoint file handling procedures are designed to provide easy manipulation of the common checkpoint file format. They are capable of reading checkpoint files produced by the hardware dump routine EDD, as well as writing checkpoint files readable by DSDI, the dump analyzer.

A checkpoint file is split into several record pairs. First there is a four word header record, which is followed by a data record. The different types are:

IMR	IOU Maintanence Registers
Inn	PPnn Memory and R-register
MMR	Memory Maintanence Registers
MEM	Central Memory
PMR	Processor Maintanence Registers
PRF	Processor Register File
PXP	Processor Exchange Package
PCS	Processor Control Store
SMC	Simulator Control Information

Not all of these are currently handled by the checkpoint routines, although the procedure to read the header records recognizes all of them. There is currently no order necessary for the header/data record pairs.

3.2.11.1 Copy_Central_Memory

This routine writes the Central Memory header and data records to the checkpoint file. The input file must be in the packed format, with 15 C180 words packed into 16 C170 words.

```
*callc pxiotyp
*callc osdstat

  PROCEDURE [XREF] ckp$copy_central_memory ALIAS 'zckpccm' (inf: file;
    checkpoint_number: 0 .. 99;
    first_word_address: 0 .. 0ffffffff(16);
    length: 0 .. 0ffffffff(16);
    VAR outf: file;
    VAR status: ost$status);
```

3.2.11.2 Pack_Central_Memory

This routine writes the Central Memory header and data records to the checkpoint file. The input file must be in the 32 in 60 bit format, that is, with the upper 32 bits in one word, and the lower 32 bits in the next.

```
*calic pxiotyp
*calic osdstat

  PROCEDURE [XREF] ckp$pack_central_memory ALIAS 'zckppcm' (inf: file;
    checkpoint_number: 0 .. 99;
    first_word_address: 0 .. 0ffffffff(16);
    length: 0 .. 0ffffffff(16);
    VAR outf: file;
    VAR status: ost$status);
```

3.2.11.3 Read_Central_Memory

This routine reads the Central Memory record of a checkpoint file, and writes it out in the packed format to 'outf'. It will also read those checkpoint files produced by the EDD hardware dump routines.

```
*calic pxiotyp
*calic osdstat

  PROCEDURE [XREF] ckp$read_central_memory ALIAS 'zckprcm' (inf: file;
    length: 0 .. 0ffffff(16);
    outf: file;
    status: ost$status);
```

3.2.11.4 Read_Header_Record

This routine reads the header records on a checkpoint file, and returns the information in the case variant record 'header_record'.

```
*callc pxiotyp
*callc zckthrc
*callc osdstat

  PROCEDURE [XREF] ckp$read_header_record ALIAS 'zckprhr' (inf: file;
    VAR header_record: ckt$header_record;
    VAR reached_eoi: boolean;
    VAR status: ost$status);
```

3.2.11.5 Read_PP_Memory

This routine reads the PP memory record, and returns the memory contents and the R-register.

```
*calic pxiotyp
*calic osdstat

  PROCEDURE [XREF] ckp$read_pp_memory ALIAS 'zckprpm' (inf: file;
    VAR r_reg: integer;
    VAR pp_memory: packed array [0 .. 16383] OF 0 .. 0f(16);
    VAR status: ost$status);
```

3.2.11.6 Read_Processor_Registers

This routine reads the Processor Maintanence Registers record of the checkpoint file and returns the registers in 'processor_registers'.

```
*callc pxiotyp
*callc zcktpmr
*callc osdstat

  PROCEDURE [XREF] ckp$read_processor_registers ALIAS 'zckprpr' (inf: file;
    VAR processor_registers: ckt$processor_registers;
    VAR status: ost$status);
```

3.2.11.7 Skip_Data_Record

This routine is for skipping data records which are not needed. It will not work on the PP and Central memory records of an EDD dump tape.

```
*calic pxiotyp

  PROCEDURE [XREF] ckp$skip_data_record ALIAS "zckpsdr" (inf: file);
```

3.2.11.8 Write_PP_Memory

This routine writes out the PP Memory header and data record to a checkpoint file.

```
*calic pxiotyp

  PROCEDURE [XREF] ckp$write_pp_memory ALIAS 'zckpwpm' (pp_number: 0 .. 19;
    r_reg: integer;
    checkpoint_number: 0 .. 99;
    pp_memory: packed array [0 .. 16383] OF 0 .. 0f(16);
    outf: file);
```

3.2.11.9 Write_Processor_Registers

This routine writes out the Processor Maintenance Registers to a checkpoint file.

```
*calic zcktpmr
*calic pxiotyp

  PROCEDURE [XREF] ckp$write_processor_registers ALIAS 'zckpwpr'
    (processor_registers: ckt$processor_registers;
    checkpoint_number: 0 .. 99;
    VAR outf: file);
```

3.3 APPENDIX OF COMMON *CALLC DECKS

The following is an alphabetic list of contents of the common decks referenced by *callc within the procedure interfaces described previously.

```
*callc osdstr

{ OSDSTAT   Definition of the NOS/180 request status record }

  CONST
    osc$max_condition = 999999,
    osc$status_parameter_delimiter = "=";

  TYPE
    ost$status_condition = 0 .. osc$max_condition,
    ost$status = record
      case normal: boolean of
      =FALSE=
        identifier: string (2),
        condition: ost$status_condition,
        text: ost$string,
      casend,
    recend;



{ZCKTCWD}

{ZCKTCWD    Defines the format of CYBER 180 words}

  TYPE
    ckt$cyber_180_word = record
      left: 0 .. 0ffffffff(16),
      right: 0 .. 0ffffffff(16),
    recend;



{ZCKTHRC}

{ZCKTHRC    Defines the header record which returns the header information}
*callc zckthrt
```

```
  TYPE
    ckt$header_record = record
      checkpoint_number: 0 .. 99,
      case header_type: ckt$header_record_types of
      = ckc$mem =
        first_word_address: 0 .. 0ffffffff(16),
        length: 0 .. 0ffffffff(16),
      = ckc$iom =
        pp_number: 0 .. 99,
      = ckc$pcs =
        no_of_128_bit_regs: 0 .. 01fffff(16),
      = ckc$unknown =
        code: 0 .. 03ffff(16),
      casend,
    recend;



{ZCKTHRT}

{ ZCKTHRT    Defines the various types of header records}

  TYPE
    ckt$header_record_types = (ckc$pmr, ckc$mmr, ckc$mem, ckc$imr, ckc$iom,
      ckc$smc, ckc$prf, ckc$pxp, ckc$pcs, ckc$unknown);



{ZCKTPMR}

{ZCKTPMR    Defines the Processor Maintenance Registers}
*callc zcktcwd

  TYPE
    ckt$processor_registers = record
      jps: 0 .. 0ffffffff(16),
      mps: 0 .. 0ffffffff(16),
      pta: 0 .. 0ffffffff(16),
      ptl: 0 .. 0ff(16),
      psm: 0 .. 07f(16),
      eid: 0 .. 0ffffffff(16),
      sit: 0 .. 0ffffffff(16),
      pid: 0 .. 0ff(16),
      ptm: ckt$cyber_180_word,
      pfs: ckt$cyber_180_word,
      dec: ckt$cyber_180_word,
      vmcl: 0 .. 0ffffffff(16),
```

```
      ss: ckt$cyber_180_word,
      oi: ckt$cyber_180_word,
    recend;



*callc osdname

{ ZOSTNAM    Defines names used by SCL. }

  CONST
    osc$max_name_length = osc$max_name_size,
    osc$max_nos170_name_length = 7;

  TYPE
    ost$name_types = (clc$nos170_name, clc$short_name, clc$long_name),
    ost$name_length = 1 .. osc$max_name_length,
    ost$nos170_name = string (osc$max_nos170_name_length),
    ost$name_descriptor = record
      typ: ost$name_types,
      length: ost$name_length,
      str: ost$name,
    recend;



*callc osdstr

{ ZOSTSTR    Defines the bounds of strings used by SCL. }

  CONST
    osc$max_string_length = osc$max_string_size;

  TYPE
    ost$string_length = 0 .. osc$max_string_length;



*callc zuttdcn

{ ZN7TFET    Type definition for NOS File Environment Table (FET). }

?? fmt ( format := off ) ??

  TYPE                          { NOS File Environment Table (FET) }
    n7t$fet = PACKED RECORD
```

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.3 APPENDIX OF COMMON *CALLC DECKS

```
      CASE filename  : utt$dc_name OF                  { fet +  0 }

      = 0 = { variation used to clear the n7t$fet }
        fet0      : 0 .. 3FFFF(16),
        fet1_22   : ARRAY[1 .. 22] OF INTEGER,

      = 1 = { main variation, good for most things }
        level_number          : 0 .. 0F(16),
        abnormal_termination  : 0 .. 0F(16),
        eoi                   : BOOLEAN,
        request_code          : 0 .. 7F(16),
        binary_operation      : BOOLEAN,
        completed             : BOOLEAN,
        device_type           : 0 .. 0FFF(16),       { fet +  1 }
        random                : BOOLEAN,
        fill0                 : SET OF 1 .. 1,
        user_processing       : BOOLEAN,
        error_processing      : BOOLEAN,
        fill1                 : SET OF 1 .. 20,
        extension_length      : 0 .. 3F(16),
        first                 : ^CELL,
        fill2                 : SET OF 1 .. 42,      { fet +  2 }
        next_in               : ^CELL,
        fill3                 : SET OF 1 .. 42,      { fet +  3 }
        next_out              : ^CELL,
        fntptr                : 0 .. 0FFF(16),       { fet +  4 }
        fill4                 : SET OF 1 .. 12,
        pru_size              : 0 .. 3FFFF(16),
        limit                 : ^CELL,
        fill5                 : SET OF 1 .. 12,      { fet +  5 }
        fwa_ws                : ^CELL,
        fill6                 : SET OF 1 .. 12,
        lwa1_ws               : ^CELL,
        cri                   : 0 .. 3FFFFFFF(16),   { fet +  6 }
        rw                    : BOOLEAN,
        rr                    : 0 .. 1FFFFFFF(16),
        fill7                 : SET OF 1 .. 24,      { fet +  7 }
        index_length          : 0 .. 3FFFF(16),
        fwa_index             : ^CELL,
        pfn                   : utt$dc_name,         { fet +  8 }
        fill8                 : SET OF 1 .. 5,
        fa                    : BOOLEAN,
        file_category         : 0 .. 3F(16),
        file_mode             : 0 .. 3F(16),
        optional_un           : utt$dc_name,         { fet +  9 }
        space                 : 0 .. 3FFFF(16),
        file_password         : utt$dc_name,         { fet + 10 }
        erad                  : ^CELL,
```

```
    user_cw             : INTEGER,                  { fet + 11 }
    packname            : utt$dc_name,              { fet + 12 }
    fill9               : SET OF 1 .. 6,
    unit                : 0 .. 0FFF(16),
    new_file_name       : utt$dc_name,              { fet + 13 }
    fill10              : SET OF 1 .. 18,

  = 2 = { variation used for PASCAL-X IO file descriptor }
    fill11                  : SET OF 1 .. 18,
    fill12                  : SET OF 1 .. 42,       { fet +  1 }
    first_as_integer        : 0 .. 3FFFF(16),
    fill13                  : SET OF 1 .. 42,       { fet +  2 }
    next_in_as_integer      : 0 .. 3FFFF(16),
    fill14                  : SET OF 1 .. 42,       { fet +  3 }
    next_out_as_integer     : 0 .. 3FFFF(16),
    fill15                  : SET OF 1 .. 42,       { fet +  4 }
    limit_as_integer        : 0 .. 3FFFF(16),
    direct  : RECORD
      current_page  : INTEGER,                      { fet +  5 }
      cri_rw_rr     : INTEGER,                      { fet +  6 }
      current_word  : INTEGER,                      { fet +  7 }
      record_length : INTEGER,                      { fet +  8 }
      file_length   : INTEGER,                      { fet +  9 }
      last_page     : INTEGER,                      { fet + 10 }
    RECEND,
    reserved1   : INTEGER,                          { fet + 11 }
    reserved2   : INTEGER,                          { fet + 12 }
    legible  : RECORD
      column          : INTEGER,                    { fet + 13 }
      remaining_chars : INTEGER,                    { fet + 14 }
      string_ptr      : INTEGER,                    { fet + 15 }
      buffer          : INTEGER,                    { fet + 16 }
      codeset         : ( ascii64#,                 { fet + 17 }
                          ascii612#,
                          ascii#     ),
    RECEND,
    print  : RECORD
      limit     : INTEGER,                        { fet + 18 }
      line      : INTEGER,                        { fet + 19 }
      page_num  : INTEGER,                        { fet + 20 }
      page_proc : ^PROCEDURE  (                   { fet + 21 }
                    print_file   : ^CELL;
                    next_page_#  : INTEGER ),
    RECEND,
    reserved3   : INTEGER,                          { fet + 22 }

  = 3 = { variation used for LFM and PFM interfacing }
    response_code     : 0 .. 0FF(16),
```

```
      fill16           : SET OF 1 .. 10,
      not_mass_storage : BOOLEAN,                    { fet +  1 }
      fill17           : SET OF 1 .. 59,
      fet2_4           : ARRAY[2 .. 4] OF INTEGER,
      fnt  : PACKED RECORD                           { fet +  5 }
        lfn            : utt$dc_name,
        fill18         : SET OF 1 .. 1,
        extend_only    : BOOLEAN,
        alter_only     : BOOLEAN,
        execute_only   : BOOLEAN,
        fill19         : SET OF 1 .. 1,
        write_lockout  : BOOLEAN,
        file_type      : 0 .. 3F(16),
        fill20         : SET OF 1 .. 1,
        control_point  : 0 .. 1F(16),
      RECEND,
      fst  : PACKED RECORD                           { fet +  6 }
        id_code                : 0 .. 3F(16),
        equipment_number       : 0 .. 3F(16),
        first_track            : 0 .. 0FFF(16),
        current_track          : 0 .. 0FFF(16),
        current_sector         : 0 .. 0FFF(16),
        fill21                 : SET OF 1 .. 3,
        file_opened            : BOOLEAN,
        file_written_since_opened  : BOOLEAN,
        file_written           : BOOLEAN,
        fill22                 : SET OF 1 .. 2,
        write_read_status      : 0 .. 3,
        last_operation_was_write   : BOOLEAN,
        busy                   : BOOLEAN,
      RECEND,
      fet7  : INTEGER,                               { fet +  7 }
      getfnt  : PACKED RECORD                        { fet +  8 }
        nf      : 0 .. 0FFF(16),
        fill23  : SET OF 1 .. 10,
        loft    : BOOLEAN,
        syft    : BOOLEAN,
        faft    : BOOLEAN,
        pmft    : BOOLEAN,
        ptft    : BOOLEAN,
        lift    : BOOLEAN,
        fill24  : SET OF 1 .. 3,
        teft    : BOOLEAN,
        phft    : BOOLEAN,
        prft    : BOOLEAN,
        roft    : BOOLEAN,
        inft    : BOOLEAN,
        fill25  : SET OF 1 .. 3,
```

```
          cb       : 0 .. 7,
          ta       : ^CELL,
        RECEND,

      = 4 = { variation used for LFM RENAME and ACCSF functions }
        fill26                 : SET OF 1 .. 18,
        fet1_5                 : ARRAY[1 .. 5] OF INTEGER,
        new_lfn                : utt$dc_name,        { fet +  6 }
        old_statement_count    : 0 .. 3FFFF(16),

      = 5 = { variation used for LFM PSCSF function }
        fill27                 : SET OF 1 .. 18,
        fill28                 : ARRAY[1 .. 5] OF INTEGER,
        fill29                 : SET OF 1 .. 12,     { fet +  6 }
        new_statement_count    : 0 .. 0FFFFFF(16),
        new_word_count         : 0 .. 0FFFFFF(16),

      = 6 .. 3FFFFFFFFF(16) = { 'unused' variations }
        ,
      CASEND,
    RECEND;

?? fmt ( format := on ) ??



{ ZN7TTSR    Contains type definition of terminal status. }

  TYPE
    n7t$terminal_status = packed record
      tid: 0 .. 3fffffffff(16),
      sys: 0 .. 3f(16),
      tn: 0 .. 0fff(16),
      fill1: set of 1 .. 24,
      int: ^cell,
      fill2: set of 1 .. 6,
      fill3: set of 1 .. 7,
      tape_mode: boolean,
      duplex: boolean,
      cset: boolean,
      init_cset: boolean,
      parity: boolean,
    recend;
```

3.0 MISCELLANEOUS ROUTINES INTERFACES
3.3 APPENDIX OF COMMON *CALLC DECKS

```
{ ZUTTDCN    Type definition for display code name. }

  TYPE
    utt$dc_name = 0 .. 3fffffffff(16);



*callc zuttdcn

{ ZUTTDNV    Type definition for display code name and value. }

  TYPE
    utt$dc_name_and_value = packed record
      dc_name: utt$dc_name,
      value: - 1ffff(16) .. 1ffff(16),
    recend;
```

Table of Contents

1.0 INTRODUCTION 1-1
1.1 SCOPE OF DOCUMENT 1-2
1.2 ASSOCIATED DOCUMENTS 1-3
1.3 NAMING CONVENTIONS 1-4
1.4 DISCLAIMER RELATED TO NOS 170 1-6
1.5 MISCELLANEOUS ROUTINES USAGE 1-7

2.0 MISCELLANEOUS ROUTINES DESCRIPTION 2-1
2.1 OBJECTIVES OF MISCELLANEOUS ROUTINES 2-2
2.2 PHILOSOPHY OF MISCELLANEOUS ROUTINES 2-3

3.0 MISCELLANEOUS ROUTINES INTERFACES 3-1
3.1 DESCRIPTION 3-1
3.2 PROCEDURES 3-2
3.2.1 DATA CONVERSION PROCEDURES 3-2
3.2.1.1 Capitalize String 3-2
3.2.1.2 Display Code Name to CYBIL String 3-3
3.2.1.3 CYBIL String to Display Code Name 3-4
3.2.1.4 CYBIL String to Display Code File Name 3-5
3.2.1.5 CYBIL String to Display Code String 3-6
3.2.1.6 Display Code String to CYBIL String 3-7
3.2.1.7 Integer to String 3-8
3.2.1.8 Integer to Right Justified String 3-9
3.2.1.9 String to Integer 3-10
3.2.1.10 Character Translation (Conversion) Structure 3-11
3.2.1.11 Word to Hexadecimal String 3-12
3.2.1.12 Word to Octal String 3-13
3.2.2 FILE PROCEDURES 3-14
3.2.2.1 Acquire a File 3-14
3.2.2.2 File Assigned to Job? 3-15
3.2.2.3 Return a File 3-16
3.2.2.4 Rewind a File 3-17
3.2.2.5 Message About NOS 170 Permenant File 3-18
3.2.2.6 Localize File 3-19
3.2.2.7 Set Record Type 3-20
3.2.2.8 Is File Writable? 3-21
3.2.2.9 Get Directory Record from Binary File 3-22
3.2.2.10 Assign Legible File to Terminal 3-23
3.2.3 SYSTEM UTILITY PROCEDURES 3-24
3.2.3.1 Current Date 3-24
3.2.3.2 Current Time 3-25
3.2.3.3 Get NOS 170 Control Statement Arguments 3-26
3.2.3.4 Program's Control Statement as CYBIL String 3-27
3.2.3.5 Get Current User Name 3-28
3.2.3.6 Issue Dayfile Message 3-29
3.2.3.7 Determine If Job Origin Is Batch 3-30
3.2.4 CYBIL PROGRAM PROCEDURES 3-31
3.2.4.1 Initiate CYBIL Program Environment 3-31
3.2.4.2 Terminate a CYBIL Program 3-32

3.2.4.3 Terminate a CYBIL Program with Generated Message 3-33
3.2.4.4 End CYBIL Program 3-34
3.2.4.5 Abort CYBIL Program 3-35
3.2.4.6 CYBIL to Compass Interface 3-36
3.2.5 STRING AND CHARACTER PROCEDURES 3-38
3.2.5.1 Compare CYBIL Strings 3-38
3.2.5.2 Build Display Code String Pointer 3-39
3.2.5.3 Get Display Code Character From String 3-40
3.2.5.4 Insert Display Code Character 3-41
3.2.6 POINTER MANIPULATION PROCEDURES 3-42
3.2.6.1 Offset of Pointer From Base 3-42
3.2.6.2 Compute Pointer From Offset 3-43
3.2.7 CYBIL OVERLAY LOADING 3-44
3.2.7.1 Overlay Structures 3-44
3.2.7.2 Load Overlay 3-45
3.2.7.3 Create Overlay Tree Structure (SES.BOVLAY) 3-48
3.2.8 USER DETECTION OF TERMINAL INTERRUPT CONDITION 3-50
3.2.8.1 Interruptable Condition Request Codes 3-50
3.2.8.2 Initialize Terminal Interrupt Detection 3-51
3.2.8.3 Was Terminal Interrupt Detected 3-52
3.2.8.4 Ask for Direction 3-53
3.2.9 GENERAL PROCEDURES 3-54
3.2.9.1 Generate Unique Alphanumeric Strings 3-54
3.2.10 CYBIL TO NOS 170 PROCEDURES 3-55
3.2.10.1 NOS 170 Combined Input Output (CIO) Request 3-57
3.2.10.2 NOS 170 Control Point Manager (CPM) 3-59
3.2.10.3 NOS 170 Local File Manager (LFM) 3-63
3.2.10.4 Message to NOS 170 Dayfile 3-66
3.2.10.5 NOS 170 Permanent File Manager (PFM) 3-67
3.2.10.6 NOS 170 Recall 3-70
3.2.10.7 NOS 170 Translate Control Statement 3-71
3.2.10.8 NOS 170 Time Processor 3-72
3.2.10.9 NOS 170 Wait Not Busy 3-75
3.2.11 CHECKPOINT FILE ROUTINES 3-76
3.2.11.1 Copy Central Memory 3-77
3.2.11.2 Pack Central Memory 3-78
3.2.11.3 Read Central Memory 3-79
3.2.11.4 Read Header Record 3-80
3.2.11.5 Read PP Memory 3-81
3.2.11.6 Read Processor Registers 3-82
3.2.11.7 Skip Data Record 3-83
3.2.11.8 Write PP Memory 3-84
3.2.11.9 Write Processor Registers 3-85
3.3 APPENDIX OF COMMON *CALLC DECKS 3-86